Proceedings of the 15th International Conference on Auditory Display, Copenhagen, Denmark May 18 - 22, 2009

OVERLAY PROBLEMS FOR MUSIC AND COMBINATORICS [1,2]

J. Allali [3], P. Antoniou [4], P. Ferraro [3,5], C. S. Iliopoulos [4,6] and S. Michalakopoulos [4,7]

[3] Laboratoire Bordelais de Recherche en Informatique, Bordeaux, France

http://www.labri.fr, {julien.allali,pascal.ferraro}@labri.fr

[4] Algorithm Design Group, Department of Computer Science, King's College London, Strand, London WC2R 2LS, England

http://www.dcs.kcl.ac.uk/research/groups/adg/, pavlos.antoniou@kcl.ac.uk,{csi,spiros}@dcs.kcl.ac.uk

[5] Pacific Institute For the Mathematical Sciences, University of Calgary, Canada

[6] Digital Ecosystems & Business Intelligence Institute and Dept. of Computing, Curtin University, Perth, Australia

[1] Research partially supported by Royal Society Joint International grant between [3] and [4], 2007-2009

[2] This work has been partially sponsored by the French ANR Brasero (ANR-06-BLAN-0045) and ANR SIMBALS (JC07-188930) projects

[7] Supported by the Engineering and Physical Sciences Research Council (EPSRC) under the Doctoral Training Award (DTA)

ABSTRACT

Motivated by the identification of the musical structure of pop songs, we introduce combinatorial problems involving overlays (non-overlapping substrings) and the covering of a text t by them. We present 4 problems and suggest solutions based on string pattern matching techniques. We show that decision problems of this type can be solved using an Aho-Corasick keyword automaton. We conjecture that one general optimization problem of the type, is NP-complete and introduce a simpler, more pragmatic optimization problem. We solve the latter using suffix trees and finally, we suggest other open problems for further investigation.

1. INTRODUCTION

A useful task within the realm of Music Information Retrieval is the automatic identification of structural parts in music pieces. This has been performed manually in some cases [12], and serves as our motivation for the problems and algorithms we present in this paper. We focus on popular western music which tends to have a structure made up of the following parts: {*Intro, Verse, Chorus, Bridge, Outro*}. For example [12], The Beatles song "Strawberry Fields Forever" has the structure shown in Figure 1.

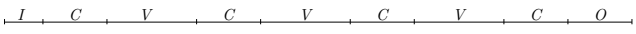

Figure 1: *"Strawberry Fields Forever" where* $I \Rightarrow Intro, V \Rightarrow Verse, C \Rightarrow Chorus, B \Rightarrow Bridge, O \Rightarrow Outro.$

Applications for music split into its structural parts are many and varied [7]. Accessing specific parts of songs, for browsing, audio thumbnailing, remixing or selectively sampling are some possible uses [11]. Skipping "unimportant" sections or facilitating a query by humming (QBH) engine [5] are others.

In the latter case, a database of "significant" parts of songs (verse and chorus) would greatly speed up the search time. This is because "users" normally use such QBH systems by humming (or inputting in another way) the more memorable part of a song i.e., the chorus or the verse [2]. Given that both the verse and the chorus are repeated patterns in the song, storing only these would reduce the storage requirements as well as the search time.

Most proposed algorithms involve signal processing [10, 11] where the music is represented and manipulated in its audio format. Instead, we follow the Mongeau and Sankoff model [9] and in true String Algorithms [3, 8] fashion represent music as a sequence of ordered pairs, with the pitch of the note as the first item and its duration as the second.

The paper is organized as follows. In Section 2 we define *overlay* and other essential concepts. In Section 3 we introduce the overlay problems. Each problem is presented and detailed solutions are investigated in the subsections 3.1 to 3.4. Finally we conclude in Section 4 and discuss further works.

2. PRELIMINARIES

A *string* is a sequence of zero or more symbols from an alphabet Σ. The set of all strings (including the empty string ε) over the alphabet Σ is denoted by Σ^*. A *text* t is a string of length $|t|$; the i^{th} symbol of t is denoted by t_i, thus $t = t_1 t_2 \dots t_{|t|}$.

A string w is a *substring* of t if $t = uwv$ for $u, v \in \Sigma^*$. A substring w of t can be represented by the pair (b, e), where $b, e \in \mathbb{N}^+$ and b is the start position and e the end position of w in t i.e., $w = t_b \dots t_e$, where $1 \leq b \leq e \leq |t|$.

A *repeat* is a non-empty substring of t that occurs at more than one position. Formally, if u occurs in t at position i i.e., $u = t_{i \dots i+|u|-1}$ then u is a repeat in t if and only if $\exists\, j \neq i, 1 \leq j \leq n$ such that $u = t_{j \dots j+|u|-1}$.

A substring w of t is of *power* m, denoted w^m when it is repeated consecutively in t, m times. Thus, if w occurs in t at i, and is of power m then w occurs at positions

$i, i + |w|, \ldots i + (m-1)|w|$ in t. For example, the empty string ε is of power 0 in any text t, and substring w of t is of power 3 if www is a substring of t.

Definition 1 (Subsequence)**.** *A* subsequence $\mathcal{S}_t$ *of* t *is a set of pairs* $\{(b_1, e_1), (b_2, e_2), \ldots, (b_m, e_m)\}$ *such that, for all* (b_i, e_i) *in* $\mathcal{S}_t$*:*

- $1 \leq b_i < e_i \leq |t|$
- $\forall (b_j, e_j) \in \mathcal{S}_t : \begin{cases} b_i \neq b_j & \text{if } i \neq j, \\ b_j > e_i & \text{if } b_j > b_i, \\ e_j < b_i & \text{if } b_j < b_i. \end{cases}$

Thus, $\mathcal{S}_t$ defines a set of non-overlapping factors of t. The length of $\mathcal{S}_t$ is equal to $\sum_{(b_i,e_i)\in\mathcal{S}_t} (e_i - b_i + 1)$ and is denoted as $\|\mathcal{S}_t\|$.

Definition 2 (Overlay)**.** *The set* $\mathcal{T} = \{w_1, w_2, \ldots, w_k\}$ *of strings* w_i *over* Σ^**, is said to be an* overlay *of text* t *if there exists a subsequence* $\mathcal{S}_t$ *of* t *such that:*

$$\forall\ (b_j, e_j) \in \mathcal{S}_t, \exists\ w_i \in \mathcal{T} \text{ such that } t_{b_j} \ldots t_{e_j} = w_i$$

In other words, each $(b_j, e_j) \in \mathcal{S}_t$ has a matching w_i of t in $\mathcal{T}$. We say that (b_j, e_j) is an *occurrence* of w_i in t, and $\mathcal{S}_t$ is an *occurrence* of $\mathcal{T}$ in t. We define the overlay *size* of $\mathcal{T}$ as follows:

$$\|\mathcal{T}\| = Max\{\|\mathcal{S}_t\| \text{ of all occurrences } \mathcal{S}_t \text{ of } \mathcal{T} \text{ in } t\} \quad (1)$$

Definition 3 (Global (Partial) Overlay)**.** *Consider a string* t *that can be entirely* covered *by an occurrence of* $\mathcal{T}$ *i.e.,* $\|\mathcal{T}\| = |t|$*.* $\mathcal{T}$ *is said to be a* global *overlay of* t*. Otherwise, when* t *cannot be entirely covered by* $\mathcal{T}$*, it is said to be a* partial *overlay.*

We call the *degree* δ of w_i the number of its occurrences in $\mathcal{S}_t$ and denote it by $\delta(w_i)$.

Example 1. *For example, for* $t = acbcaaccbbca$*,* $\mathcal{T} = \{w_1 = cb, w_2 = bca, w_3 = ac\}$ *is a global overlay of* t *(*$S_t = \{(1,2),(3,5),(6,7),(8,9),(10,12)\}$*). The degree of the substring* cb*,* $\delta(w_1)$*, is* 1*, while* $\delta(w_2) = \delta(w_3) = 2$*. It is clear that* w_2 *and* w_3 *are repeats in* t*. All the substrings* w_i *are of power* 1 *in* t*.*

Note that for $t' = cbca$*, although it can be said that* w_1 *and* w_2 *cover* t'*, we cannot exhibit a subsequence for* $\mathcal{T}$ *because the two substrings overlap in* t'*.*

3. THE PROBLEMS

There are two kinds of problems:

- Given a set of strings $\mathcal{T}$, and text t, validation that $\mathcal{T}$ is a global overlay of t (VALIDATION PROBLEMS). These are "decision" problems.
- Inferring an *optimal* (longest length) partial overlay of a text under some restrictions (INFERENCE PROBLEMS). These are "optimization" problems.

We next present 4 problems, two of each general type. We note their motivations from music and mathematics, present solutions and make a couple of conjectures.

3.1. Global Overlay Problem

Problem 1 (Global Overlay)**.** *Given text* $t \in \Sigma^*$ *of length* n*, and a set of strings* $\mathcal{T} = \{w_1, w_2, \ldots, w_k\}$ *over* Σ^**, is there a global overlay of* t *over* $\mathcal{T}$*?*

This is a validation problem. Next we present an efficient solution to this problem and conjecture that it is the fastest possible.

An outline of the algorithm:

STEP 1
Build the Aho-Corasick (AC) automaton [1], for the set of strings $\mathcal{T} = \{w_1, w_2, \ldots, w_k\}$.

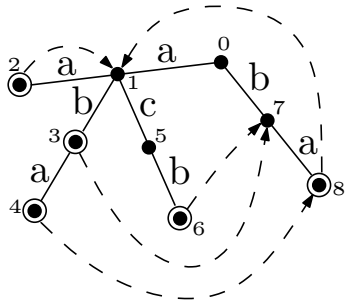

Figure 2: *Aho-Corasick automaton for* $\mathcal{T} = \{w_1 = aa, w_2 = ab, w_3 = aba, w_4 = acb, w_5 = ba\}$*. Only non-trivial failure links are shown.*

STEP 2
Trace t via the AC automaton, finding which strings w_i of $\mathcal{T}$ occur at position j of t, $\forall j$. Let $\mathcal{T}'_j = \{w_i \mid t_{j-|w_i|}..t_j\}$.

STEP 3
Assume that we've computed whether there is a valid overlay at all positions of t up to j i.e., for substring $t_1..t_j$ of t. We will now check whether there is a valid overlay at position $j+1$. Say $\mathcal{T}'_{j+1} = \{w_{i_1}, w_{i_2}, \ldots, w_{i_\ell}\} \subseteq \mathcal{T}$ occur at position $j+1$, then we check each w_{i_p}, where $p \in [1..\ell]$, until we find a overlay at position $j+1-|w_{i_p}|$ of t or report that there isn't one in $t_1..t_{j+1}$.

Example 2. *Given the string* $t = abaababaabaab$ *and the set of strings* $\mathcal{T} = \{w_1 = aa, w_2 = ab, w_3 = aba, w_4 = acb, w_5 = ba\}$ *we construct the AC automaton shown in Figure 2. The trace on* t *of the automaton is shown in Figure 3, from which we infer that* $\mathcal{T}$ *is a overlay of* t*.*

3.1.1. Complexity Analysis

Step 1 depends on the sum of the lengths of the strings, which we denote $\|\mathcal{T}\|$ i.e., $\|\mathcal{T}\| = |w_1| + |w_2| + \ldots + |w_k|$ and the size of the alphabet Σ. For fixed size alphabet it can

i	t_i	AC node	$\mathcal{T}'$	overlay at i
1	a	1	–	–
2	b	3	$\{w_2\}$	w_2
3	a	4	$\{w_3, w_5\}$	w_3
4	a	2	$\{w_1\}$	$w_2.w_1$
5	b	3	$\{w_2\}$	$w_3.w_2$
6	a	4	$\{w_3, w_5\}$	$w_3.w_3, w_2.w_1.w_5$
7	b	3	$\{w_2\}$	$w_3.w_2.w_2$
8	a	4	$\{w_3, w_5\}$	3 solutions, one is: $w_3.w_3.w_5$
9	a	2	$\{w_1\}$	$w_3.w_2.w_2.w_1$
10	b	3	$\{w_2\}$	3 solutions
11	a	4	$\{w_3, w_5\}$	4 solutions
12	a	2	$\{w_1\}$	3 solutions
13	b	3	$\{w_2\}$	4 solutions

Figure 3: *Trace of automaton on string t.*

be considered linear on the size of $\|\mathcal{T}\|$. Step 2 is linear on $|t|$.

The runtime of step 3 depends on the number of occurrences of strings w_i identified at the terminal nodes of the automaton i.e., $|\mathcal{T}_i'|$. In the worse case $\mathcal{T}' = \mathcal{T}$, so k strings must be checked at each position of t. Assuming that $\|\mathcal{T}\| < t$, which is generally the case, our algorithm runs in $O(kn)$.

We believe that there isn't a faster solution to this problem and so state the following conjecture, which we leave as an open problem.

Conjecture 1. *Problem 1 cannot be determined any faster than $O(kn)$.*

Our solution answers the question of whether a given set is a valid overlay. Can we, given the above problem also infer *all* the global overlays of t? The last column in Figure 3 shows an increasing number of possible solutions. Based on this observation we state and prove the following lemma:

Lemma 1. *The number of global overlays of a text t of length n, over a set of strings $\mathcal{T} = \{w_1, w_2, \ldots, w_k\}$ can be greater than k^n.*

Proof. Let $\mathcal{T} = \{a, a^2, a^3, \ldots, a^k\}$, and $t = a^n$. All k strings occur at position i. This implies k combinations at each position leading to k^n possible covers. □

3.2. Partial Overlay with Minimum Degree d Problem

Problem 2 (Partial Overlay with Minimum Degree d)**.** *Given text $t \in \Sigma^*$, find the set $\mathcal{T} = \{w_1, w_2, \ldots, w_k\}$ of strings over Σ^*, where the degree of each w_i is at least d, $\delta(w_i) \geq d$, which 'best' covers the string t i.e., with the largest overlay size.*

This is an inference problem. For $d = 1$, $\mathcal{T} = \{w_1\} = t$, trivially. For $d = 2$, an outline of an algorithm:

STEP 1
Find all repeated strings in t, done in linear time using a suffix tree [6, 13], for example.

STEP 2
Form combinations of these repeated strings and try to cover as much as possible of t without the strings overlapping.

For example, it may be possible to cover the whole of t with just one string. Consider the text $t = abcdabcd$ which has a global overlay $\mathcal{T} = \{w = abcd\}$. Or alternatively, we may need all of the repeated strings in t. This time consider text $t = abcddcba$ which has a global overlay $\mathcal{T} = \{w_1 = a, w_2 = b, w_3 = c, w_4 = d\}$.

Conjecture 2. *The partial overlay with minimum degree d problem is NP-complete.*

This can be proven as future works, possibly by reducing 3-SAT to Problem 2, as was done for a similar problem in [4]. In this paper we solve a simpler inference problem with an application in Music Information Retrieval in Section 3.4. But first we solve a verification problem in the next section, which also has real world applications.

3.3. Global Overlay with Powers Problem

Problem 3 (Global Overlay with Powers)**.** *Given text $t \in \Sigma^*$ of length n, and a set of strings $\mathcal{T} = \{w_1, w_2, \ldots, w_k\}$ over Σ^*, is there a global overlay of t over $\mathcal{T}$ such that every string w_i involved in the overlay is a power of at least m i.e., $t = w_{i_1}^{k_1} w_{i_2}^{k_2} \ldots w_{i_\ell}^{k_\ell}$, where $k_i \geq m$.*

This verification problem is motivated by the fact that often, the strings that represent pop songs can be broken down into substrings that repeat consecutively. This problem is also interesting from a combinatorial point of view. The solution we present is similar to that of Problem 1.

An outline of the algorithm:

STEP 1
Build the Aho-Corasick (AC) automaton for the set of strings and their powers $\mathcal{T}^{(m)} = \{w_1^m, w_1, w_2^m, w_2 \ldots, w_k^m, w_k\}$.

STEP 2
Trace t via the AC automaton, finding which strings w_i^q of $\mathcal{T}^{(m)}$ occur at position j of t, $\forall j$, where $q \in \{m, 1\}$. Let $\mathcal{T}_j'^{(m)} = \{w_i^q \mid t_{j-|w_i^q|}..t_j\}$.

STEP 3
Assume that we've computed whether there is a valid overlay at all positions of t up to j i.e., for substring $t_1..t_j$ of t. We will now check whether there is a valid overlay at position $j+1$. Say $\mathcal{T}_{j+1}'^{(m)} = \{w_{i_1}^{q_1}, w_{i_2}^{q_2}, \ldots, w_{i_\ell}^{q_\ell}\} \subseteq \mathcal{T}$ occur at position $j+1$, then we check each $w_{i_p}^{q_p}$, where $p \in [1..\ell]$, until we find a overlay at position $j + 1 - |w_{i_p}^{q_p}|$ of t, only this time, if $w_{i_p}^{q_p}$ is not a power of m but a single occurrence i.e., $q_p = 1$ then it is only valid if there is an occurrence at $j+1-|w_{i_p}^{q_p}|$ which is a power. In other words, if at position $j+1$, $\mathcal{T}_{j+1}'^{(m)}$ contains both "single" strings and powers, keep a power. If there are only single strings then the sequence is valid up to j only if there is a power up to $j - w_i$.

The runtime is the same as for Problem 1 i.e., $O(kn)$ even though the hidden constant is larger due to the extra checks for single occurrences or powers.

3.4. 2 String Partial Overlay with Minimum Degree 2 Problem

Next we describe the final problem of this paper which is motivated by the fact that pop songs are often composed of verse and chorus, which repeat at least once each and form the main body of the musical piece, see [12]. Note that if we identify two repeated substrings in musical text t which cover *most* of t, then with the help of [7] we can infer that the remaining sections are intro, outro, and bridge and solo or other meaningful labels, Figure 4.

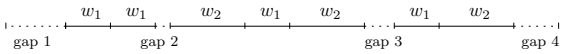

Figure 4: *By identifying verse and chorus (w_1 and w_2), the other bits (intro, outro, bridge, solo) may be inferred.*

Problem 4 (2 String Partial Overlay with Minimum Degree 2)**.** *Given a text t find the optimal partial overlay for 2 strings with each having degree at least 2, and the condition that each occurrence of the strings is present in the overlay. Formally, find $\mathcal{T} = \{w_1, w_2\}$ with largest $\|\mathcal{T}\|$ such that $\delta(w_1) \geq 2$ and $\delta(w_2) \geq 2$ and all occurrences of w_1 and w_2 in t are involved in the overlay.*

This is an inference problem. The first observation is that for $\delta(w_1)$ and $\delta(w_2)$ to be at least 2, the only substrings we need consider are the repeats of t.

An outline of the algorithm:

STEP 1
Find all repeated substrings of t. We can do this using a suffix tree.

STEP 2
Form pairs of substrings and if they give a valid partial overlay i.e., they don't overlap in any of their occurrences in t, report the length of the subsequence $\|\mathcal{S}_t\|$.

STEP 3
Find the best of the valid partial overlays by Equation 1.

Next we define our suffix tree and then present the details of the outlined algorithm.

3.4.1. Further Definitions

We call $ST(t)$ the *suffix tree* of t of length n. The tree nodes are labeled $v_0, v_1, \ldots$ with v_0 the *root* node. For each node v_i of $ST(t)$, F_{v_i} denotes the word spelt from the root to v_i i.e., it's *path label*.

A *suffix link* $s(v)$ of node v in tree $ST(t)$ is a directed edge from internal node v to internal node u such that if the path label F_v of v is $x\beta$ where x a symbol in Σ and β a string (possibly empty) in Σ^* then the path label F_u of u, is equal to β.

3.4.2. Finding the Overlay

Every repeated substring of t is encoded in the tree at the internal nodes. They are the path labels of the nodes and the prefixes of the path labels.

To find the optimal overlay, we can simply compare each pair of repeated substrings and see if they overlap and if they don't then calculate the overlay as outlined in step 2 and finally, report the optimal cover in step 3.

We observe however that many of the pairs overlap and that this information is also encoded in suffix tree $\mathcal{ST}(t)$. We apply two rules:

1. If u is an ancestor of v, then u's path label is a prefix of v's path label, and u and v overlap. (ANCESTOR RULE)

2. If u is the suffix link node of v, then u's path label is a suffix of v's path label, and thus they overlap. Further, all of u's ancestors are also suffixes of u. (SUFFIX RULE)

Let $\mathcal{L}(t)$ be the set of pairs of repeated substrings in t we need compare i.e., $\mathcal{L}(t) = \{(u, v)\}$ such that (u, v) are pairs of nodes in $\mathcal{ST}(t)$ with path labels and prefixes of path labels that don't break the Ancestor Rule or the Suffix Rule. Further, let $P(u)$ be the list of ancestors of node u in $\mathcal{ST}(t)$. We build $\mathcal{L}(t)$ in Algorithm 1. The running time depends on the number of suffix links that each node needs to traverse. Given an extreme case such as string $(ab)^n$ would require $O(n)$ iterations of the while loop, giving a total runtime of $O(nm)$, where m the number of internal nodes in $\mathcal{ST}(t)$.

Algorithm 1 Find Valid Overlay Pairs

```
1:  function FINDOVERLAYPAIRS(ST(t))
2:      L = ∅
3:      for each u ∈ ST(t) do
4:          list L_u = {all path labels and node prefixes in ST(t)}
5:          while u ≠ root do
6:              remove self from L_u                    ▷ remove u
7:              remove ancestors from L_u               ▷ remove P(u)
8:              set u to suffix link node               ▷ u ← s(u)
                                                        ▷ add u's valid pairs to L
9:          L ← L ∪ {(u, v) | v ∈ L_u}
10:     return L
```

By increasing the space complexity we can reduce the running time. At each node v_i we store data as shown in Figure 5. There are three binary lists. The *ancestor list* $P(v_i)$, represents the nodes with which v_i breaks the ancestor rule. A 1 at position j of v_i's list signifies that v_j is an ancestor of v_i. The binary list is simply constructed by the application of a logical OR ($|\ |$) thus:

$$P(v_i) \ = \ v_i \ | \ | \ P(p(v_i))$$

where v_i is a binary list that represents the node itself (e.g. node $v_2 = 01000...$) and $p(v_i)$ is the nodes parent.

It may be obvious at this point that the order in which the nodes are processed is important for this operation to give us the expected result i.e., for v_i's ancestor list to exclude (include) all ancestors of v_i. For the nodes to be processed in the correct order, we store them in a sorted balanced binary tree $B(t)$. The nodes are sorted by their depth in the suffix tree $\mathcal{ST}(t)$ so that any node v_i that is an ancestor or a suffix link of a node v_j is before v_i in $B(t)$.

The *suffix link list* $S(v_i)$, represents the nodes with which v_i breaks the suffix rule:

$$S(v_i) = P(s(v_i)) \,||\, S(s(v_i))$$

where $s(v_i)$ is v_i's suffix link. Note that neither the ancestor list nor the suffix list is *symmetric*.

The *overlay list* $C(v_i)$, represents the pairs of nodes that must be compared. Thus, it's symmetric because pair (v_i, v_j) is equal to pair (v_j, v_i) in this case. The overlay list is calculated:

$$C(v_i) \;=\; P(v_i) \,||\, S(v_i) \,||\, P^{-1}(v_i) \,||\, S^{-1}(v_i)$$

where $P^{-1}(v_i)$ is a binary list such that:

$$P^{-1}(v_i)[j] \;=\; P^{-1}(v_j)[i],\ \forall\,(v_i, v_j).$$

Similarly $S^{-1}(v_i)$ is a binary list such that:

$$S(v_i)[j] \;=\; S(v_j)[i],\ \forall\,(v_i, v_j).$$

We use Algorithm 2 to build the binary list C_{v_i} for node v_i. Each logical OR can be performed in constant time and so the algorithm takes $O(m)$ time, where m the number of internal nodes in $\mathcal{ST}(t)$.

Algorithm 2 Find Valid Overlay Pairs Fast

```
1:  function FINDOVERLAYPAIRSFAST(ST(t))
2:      v_i ← first node in B(t)
3:      while v_i not null do
4:          P(v_i) ← v_i || P(p(v_i))
5:          v_i ← next node in B(t)
6:      v_i ← first node in B(t)
7:      while v_i not null do
8:          S(v_i) ← P(s(v_i)) || S(s(v_i))
9:          v_i ← next node in B(t)
10:     v_i ← first node in B(t)
11:     while v_i not null do
12:         C(v_i) = P(v_i) || S(v_i) || P^{-1}(v_i) || S^{-1}(v_i)
13:         v_i ← next node in B(t)
14:     return C(v_i)
```

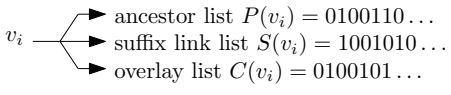

Figure 5: Node data

4. CONCLUSION AND FURTHER WORKS

Proving the conjectures we've made in this paper are our immediate targets and though it may not be interesting from a musical point of view, it is from a purely Mathematical and Computer Science point of view.

Further works include implementing the suffix tree solution to Problem 4 and testing it on real data. We believe we can achieve good percentages of identifying the full structure of pop songs i.e., verse, chorus, bridge, intro, and outro. We intend to consider the same problem but by allowing inexact repeats as well. The test results would be expected to be even better for such an algorithm and implementation given that it is common that two verses in the same song would have slight differences between them.

The solution to Problem 4 could also be improved by reducing the number of strings that need pairing even further and proving a (possibly linear) upper bound on the number of pairs. Lifting some of the restrictions is an additional improvement that can be made, in particular the one that all occurrences must be involved in the overlay.

We've presented 4 overlay related problems. These problems attempt to identify the non-overlapping occurrences of substrings of a given text t. The problems are split into optimization (inference problems) and decision (validation problems) and we suggest solutions for each one from the domain of string pattern matching.

We solve the decision problems by using Aho-Corasick keyword searching techniques. We conjecture that a validation problem is NP-complete and solve a simpler version of it in polynomial time by using a suffix tree and manipulation of binary arrays. This latter problem has direct applications in the real world since we make the case that it can be used for identifying the structure of popular western music.

5. REFERENCES

[1] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

[2] Vaino Ala-Harkonen, Johan Brunberg, Kjell Lemstrom, and Niko Mikkila. Jmir serves mozart. In *Proceedings of the 2008 Computers in Music Modeling and Retrieval Conference (CMMR 2008), Copenhagen, Denmark*, pages 148–157, May 2008.

[3] J. Allali, P. Ferraro, P. Hanna, and C. S. Iliopoulos. Local transpositions in alignment of polyphonic musical sequences. In *14th String Processing and Information Retrieval Symposium, Santiago, Chile*, 2007.

[4] R. Cole, C. S. Iliopoulos, M. Mohamed, W. F. Smyth, and L. Yang. Computing the minimum k-cover of a string. In M. Simanek, editor, *Proceedings of the 2003 Prague Stringology Conference (PSC'03)*, pages 51–64, 2003.

[5] Asif Ghias, Jonathan Logan, David Chamberlin, and Brian C. Smith. Query by humming: musical information retrieval in an audio database. In *ACM Multimedia*, pages 231–236, 1995.

[6] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[7] Paulus J. and Klapuri A. Labelling the structural parts of a music piece with markov models. In *Proceedings of the 2008 Computers in Music Modeling and Retrieval Conference (CMMR 2008), Copenhagen, Denmark*, pages 137–147, May 2008.

[8] K. Lemstrom. String matching techniques for music retrieval. *PhD Thesis, University of Helsinki, Department of Computer Science*, 2000.

[9] M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.

[10] B. S. Ong. *Structural Analysis and Segmentation of Musical Signals*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Spain, 2006.

[11] Geoffroy Peeters. Deriving musical structures from signal analysis for music audio summary generation: sequence and state approach. *Lecture Notes in Computer Science*, 2771/2004:169–185, February 2004.

[12] A. W. Pollack. 'notes on...' series. the official rec.music beatles home page. http://www.recmusicbeatles.com. 1989-2001.

[13] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, pages 249–260, 1995.